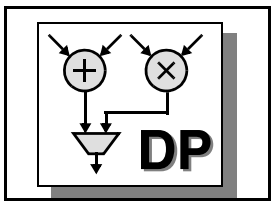

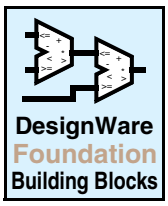

DWF_dp_blend functions

Graphics Alpha Blend

Version, STAR, and myDesignWare Subscriptions: IP Directory

Description

The DWF_dp_blend functions implement an alpha blender or linear interpolator. They blend two input pixels (arguments `x` and `y`) with a factor (argument `alpha`) and return the output pixel. All arguments and the return value are unsigned.

Table 1-1 lists the different functions, which trade off result accuracy versus circuit QoR.

Table 1-1 DWF_dp_blend Function Names

Function Name	Description
DWF_dp_blend	VHDL/Verilog basic blend (best QoR, result has error of 1 LSB)
DWF_dp_blend_exact	VHDL/Verilog exact blend (worst QoR, exact result)
DWF_dp_blend_exact2 [a]	VHDL/Verilog exact2 blend (good QoR, exact result, `alpha1` input)
DWF_dp_blend2	VHDL/Verilog basic blend without rounding/truncation
DWF_dp_blend2_exact2 [b]	VHDL/Verilog exact2 blend without rounding/truncation

a. Compatible with the MCL function `gfxBlend` from Module Compiler (MC).
b. Compatible with the MCL function `gfxBlend2` from Module Compiler (MC).

Table 1-2 Argument Descriptions

Argument Name	Type	Width / Values	Description
x	Vector	width	Input pixel
y	Vector	width	Input pixel
alpha	Vector	alpha_width	Blend factor
alpha1 [a]	Bit	1	Blend factor 1 indicator
DWF_dp_blend DWF_dp_blend_exact DWF_dp_blend_exact2	Vector	width	Return value
DWF_dp_blend2 DWF_dp_blend2_exact2	Vector	width+alpha_width	Return value

a. Only for functions DWF_dp_blend_exact2 and DWF_dp_blend2_exact2.

Table 1-3 Parameter Descriptions (Verilog)

Parameter	Values	Description
width	≥ 2	Word length of inputs `x` and `y`
alpha_width	≥ 2	Word length of input `alpha`

Verilog Include File: `DW_dp_blend_function.inc`

Functional Description

A graphics alpha blender, or linear interpolator, implements the programmable blend of two input pixels to form the output pixel. The blend factor α is a real value in the range of 0 to 1.

$$Z = \alpha \cdot X + (1 - \alpha) \cdot Y, 0 \leq \alpha \leq 1$$

In a hardware implementation, it is not feasible to have a real α value for QoR reasons. Therefore α is coded as an integer input `alpha`.

Basic Blend (`DWF_dp_blend`)

In the basic blend implementation, α is coded as an integer `alpha` with `alpha_width` bits, which is interpreted as the fraction of a fixed-point number (α = `0.alpha` = `alpha` / 2^{alpha_width}). Hardware implementation is efficient since the lower `alpha_width` bits of the intermediate result can simply be truncated (divided by 2^{alpha_width}). $2^{alpha\_width-1}$ is added for rounding before truncation (corresponds to adding 0.5 to the final result).

```
z = DWF_dp_blend (x, y, alpha)
```

$$z = (alpha * x + \sim alpha * y + 2^{alpha\_width-1}) / 2^{alpha\_width}$$

However, the value α = 1 cannot be represented by `alpha`. Therefore, the blend result is not exact and can have an error of at most 1 LSB.

Exact Blend (`DWF_dp_blend_exact`)

The entire range of α (including value 1) can be represented if `alpha` (or the intermediate result) is divided by $2^{alpha\_width}-1$ (α = `alpha` / ($2^{alpha\_width}-1$)), see also Table 1-4 on page 3. The blend result is now exact. However, division by $2^{alpha\_width}-1$ is not trivial and more expensive in hardware, resulting in worse QoR. No divider circuit is used in the synthesis implementation though.

```
z = DWF_dp_blend_exact (x, y, alpha)
```

$$z = (alpha * x + \sim alpha * y + 2^{alpha\_width-1}-1) / (2^{alpha\_width}-1)$$

Exact2 Blend (`DWF_dp_blend_exact2`)

Another possibility to calculate an exact blend is to use an extra input bit `alpha1` to represent α (α = (`alpha` + `alpha1`) / 2^{alpha_width}). The value α = 1 is then obtained by setting `alpha1 = 1` and `alpha = "1...11"`. For all other values of α set `alpha1 = 0`. The intermediate result can now be

truncated (divided by 2^{alpha_width}). QoR is therefore better than for the exact blend, but coding of α using two inputs `alpha` and `alpha1` might not be as convenient.

```
z = DWF_dp_blend_exact2 (x, y, alpha, alpha1)
```

$$z = (alpha * x + \sim alpha * y + ((alpha1 == 1)\ ?\ x : y) + 2^{alpha\_width-1}) / 2^{alpha\_width}$$

Basic Blend without Rounding/Truncation (`DWF_dp_blend2`)

The basic blend is available without the rounding and truncation. This is useful when a different or no rounding / truncation is desired or if it is done elsewhere in the code.

```
z = DWF_dp_blend2 (x, y, alpha)
```

```
z = alpha * x + ~alpha * y
```

Exact2 Blend without Rounding/Truncation (`DWF_dp_blend2_exact2`)

The exact2 blend is available without the rounding and truncation. This is useful when a different or no rounding / truncation is desired or if it is done elsewhere in the code.

```
z = DWF_dp_blend2_exact2 (x, y, alpha, alpha1)
```

```
z = alpha * x + ~alpha * y + ((alpha1 == 1) ? x : y)
```

Table 1-4 Example α values (8-bit)

alpha1	alpha	α for Basic Blend	α for Exact Blend	α for Exact2 Blend
0	00000000	0/256 (= 0.0)	0/255 (= 0.0)	0/256 (= 0.0)
0	00000001	1/256	1/255	1/256
...	...	...	...	...
0	10000000	128/256 (= 0.5)	128/255	128/256 (= 0.5)
...	...	...	...	...
0	11111111	255/256 (< 1.0)	255/255 (= 1.0)	255/256 (< 1.0)
1	11111111			256/256 (= 1.0)

For more information about the DesignWare datapath functions, refer to the topic titled Arithmetic – Datapath Functions Overview.

Related Topics

- Arithmetic – Datapath Functions Overview
- DesignWare Building Block IP User Guide

VHDL Example

```
library IEEE, DWARE;
use IEEE.std_logic_1164.all;
use IEEE.numeric_std.all;
use DWARE.DW_dp_functions.all;
-- DWARE.DW_dp_functions_arith package if IEEE.std_logic_arith is used

entity DWF_dp_blend_test is
  port (x_pict, y_pict         : in  unsigned(15 downto 0);
        x_text, y_text, z_text : in  unsigned(15 downto 0);
        alpha                  : in  unsigned( 7 downto 0);
        z                      : out unsigned(15 downto 0));
end DWF_dp_blend_test;

architecture rtl of DWF_dp_blend_test is
begin
  z <= DWF_dp_blend (x_pict + x_text, y_pict + y_text, alpha) + z_text;
end rtl;
```

Verilog Example

```
module DWF_dp_blend_test (x_pict, y_pict, x_text, y_text, z_text, alpha, z);

  input  [15:0] x_pict, y_pict;
  input  [15:0] x_text, y_text, z_text;
  input   [7:0] alpha;
  output [15:0] z;

  // Passes the parameters to the function
  parameter width       = 16;
  parameter alpha_width = 8;

  // add "$SYNOPSYS/dw/sim_ver" to the search path for simulation
  `include "DW_dp_blend_function.inc"

  assign z = DWF_dp_blend (x_pict + x_text, y_pict + y_text, alpha) + z_text;

endmodule
```

Copyright Notice and Proprietary Information

Destination Control Statement

All technical data contained in this publication is subject to the export control laws of the United States of America. Disclosure to nationals of other countries contrary to United States law is prohibited. It is the reader's responsibility to determine the applicable regulations and to comply with them.

Disclaimer

SYNOPSYS, INC., AND ITS LICENSORS MAKE NO WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, WITH REGARD TO THIS MATERIAL, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE.

Trademarks

Synopsys and certain Synopsys product names are trademarks of Synopsys, as set forth at https://www.synopsys.com/company/legal/trademarks-brands.html.

All other product or company names may be trademarks of their respective owners.

Free and Open-Source Software Licensing Notices

If applicable, Free and Open-Source Software (FOSS) licensing notices are available in the product installation.

Third-Party Links

Any links to third-party websites included in this document are for your convenience only. Synopsys does not endorse and is not responsible for such websites and their practices, including privacy practices, availability, and content.

Synopsys, Inc.
www.synopsys.com